LA-UR-02-6987

Title: À la carte: A Simulation Framework for Extreme-Scale Hardware Architectures

Author(s): Kathryn Berkbigler, Brian Bush, Kei Davis, Nicholas Moss, Steve Smith, Thomas P. Caudell, Kenneth L. Summers, and Cheng Zhou

Submitted to: IASTED International Conference
on Modelling and Simulation
24-26 February 2003
Palm Springs, California

Los Alamos
NATIONAL LABORATORY

À la carte: A Simulation Framework for Extreme-Scale Hardware Architectures

Kathryn Berkbigler, Brian Bush, Kei Davis, Nicholas Moss, and Steve Smith
Los Alamos National Laboratory
Los Alamos, NM 87545
email: {kpb, bwb, kei, nickm, sas}@lanl.gov

Thomas P. Caudell, Kenneth L. Summers, and Cheng Zhou
Albuquerque High Performance Computing Center
University of New Mexico
Albuquerque, NM 87131
email: tpc@eece.unm.edu, {summers, czhou}@ahpcc.unm.edu

ABSTRACT
We outline *à la carte*, an approach for simulating computing architectures applicable to extreme-scale systems (thousands of processors) and to advanced, novel architectural configurations. Our component-based design allows for the seamless assembly of architectures from representations of workload, processor, network interface, switches, etc., with disparate resolutions, into an integrated simulation model. This accommodates different case studies that may require different levels of fidelity in various parts of a system. The current implementation includes low- and medium-fidelity models of the network and low-fidelity and direct execution models of the workload. It supports studies of both simulation performance and scaling, and the properties of the simulated system themselves.

KEY WORDS
Discrete event simulation, object oriented implementation, hardware architecture, network.

1 Introduction

The magnitude of the scientific computations targeted by the US DOE ASCI project requires unprecedented computational power, and sufficient bandwidth to enable remote, real-time interaction with the compute servers. To facilitate these computations ASCI plans to deploy massive computing platforms, possibly consisting of tens of thousands of processors, capable of achieving 10-100 TeraOPS, with WAN connectivity from these to distant sites.

Better hardware design and lower development costs require performance evaluation, analysis, and modeling of parallel applications and architectures, and in particular *predictive* capability. Performance studies are routinely used to select the best architecture or platform for a given application, select the best algorithm for solving a particular problem, and to study scalability with respect to problem and platform size. Evaluating and analyzing performance is challenging primarily because of the large number of components making up such systems and the complex dynamic interactions between them. For systems of ASCI-proposed size and complexity *simulation* is the predictive tool of choice, though simulation may be fruitfully augmented by analytical and statistical analysis.

The simulation environment under development is intended to allow (i) exploration of hardware/architecture design space; (ii) exploration of algorithm/implementation space both at the application level (e.g. data distribution and communication) and the system level (e.g. scheduling, routing, and load balancing); (iii) determining how application performance will scale with the number of processors or other components; (iv) analysis of the tradeoffs between performance and cost; and, (v) testing and validating analytical models of computation and communication.

The *à la carte* project [1] seeks to develop a simulation-based analysis tool for evaluating massively-parallel computing platforms including current and future ASCI-scale systems. Such a tool will provide a means to analyze and optimize the current systems and applications as well as influence the design and development of next-generation high-performance computers. Hence our general goal is to design and implement a *flexible* and *modular* simulation framework for design and analysis of extreme-scale parallel and distributed computing systems, and as an ongoing part of this process to validate the accuracy of results characterized by any particular model. An intermediate goal is to model, and validate the model of, the ASCI Q machine [2] with a realistic ASCI workload.

The simulation system should

- be scalable to model systems comprising 10,000 processors or more;
- allow components to be represented with arbitrary degrees of fidelity, in terms of both structure (e.g. comprising distinct subcomponents) and timing;

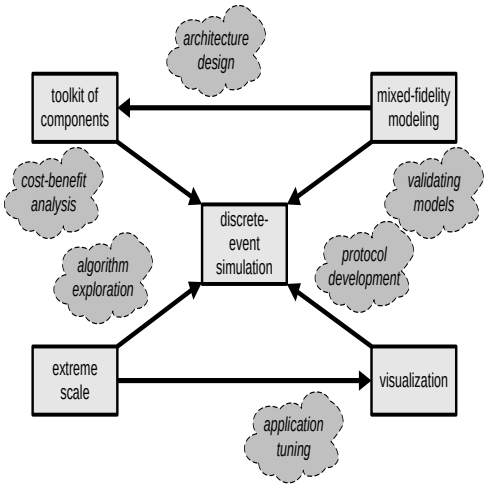

Figure 1. Goals and applications of the *à la carte* project: The boxes represent the project goals and the "clouds" represent applications for the simulation and analysis tool.

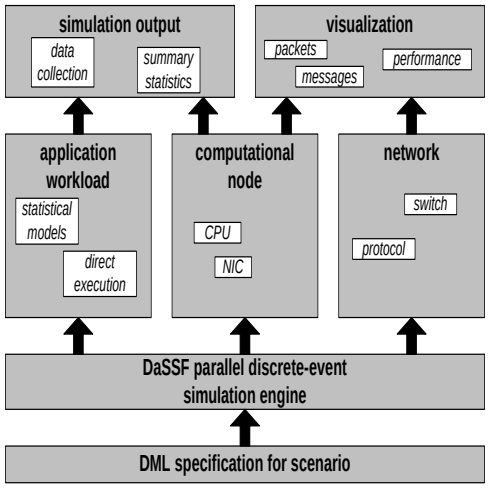

Figure 2. The architecture of the *à la carte* simulator: Simulation scenarios are represented using DML and managed by the DaSSF simulation engine. The application workloads, computational nodes, and networks are represented by software components that are assembled and connected according to the particular scenario being simulated. The results of the simulation may be studied visually, statistically, or in detail.

- allow arbitrary (meaningful) configuration of components;
- be genuinely portable across platforms ranging from single-processor workstations to clusters of SMPs.

Figure 1 graphically illustrates these goals and the applications they support. These requirements suggest factoring the simulator into three parts: component descriptions, configuration descriptions, and an underlying, reasonably generic, and reasonably light-weight simulation system with which all porting issues are associated. An object-oriented approach facilitates these goals.

It is clear that simulating systems of the size and complexity that we envision will require the use of parallel simulation; ideally the simulation system would support distributed and shared memory simultaneously, i.e. fully exploit clusters of SMPs. Furthermore, the parallel simulation substrate must support composition of simulations and be very efficient in its implementation. Several parallel discrete event simulation environments exist that may be used for computer architecture simulation. References [3] and [4] contain surveys of languages and libraries for parallel discrete event simulation. We concluded that a conservative synchronization scheme would have the best chance of success for this application, and describe our selected simulation environment in the following section.

2 Simulator Architecture

Our basic approach relies on an iterative development process for constructing components of appropriate fidelities and integrating them into a portable and efficient parallel discrete event simulation that is scalable to thousands of (simulated) computational nodes. Components may be processors, switches, network interfaces, or application workloads, for example. Studies of hardware architectures are made by running our simulation for a particular aggregate system composed of these components. Figure 2 illustrates the architecture of our simulator.

We chose a portable, conservative synchronization engine, the Dartmouth Scalable Simulation Framework (DaSSF) [5, 6], Dartmouth College's implementation of the Scalable Simulation Framework API [7], for the handling of discrete events. DaSSF manages the synchronization, scheduling, and delivery of events in the simulation; it has a lean C++ API and supports both shared-memory and distributed-memory parallelism. We use the Domain Modeling Language (DML) [8] to specify the architecture and workload to be simulated. DML allows one to easily construct libraries of reusable component specifications. The lower two levels of the architecture in Figure 2 comprise DML and DaSSF.

Our component-based design (represented by the middle layer in Figure 2) allows for the seamless assembly of architectures from representations of workloads, processors, network interfaces, switches, etc., with disparate resolutions, into an integrated simulation model. Components of different fidelities may be mixed and matched to construct a model with the appropriate level of detail for a particular study. We are focusing on the development of a simulation capability that scales to tens of thousands of processors and that can execute on a wide variety of computing platforms.

We achieve this seamless integration of disparate fidelities, and flexibility in constructing models, by exploit-

ing the design of DaSSF and using well-known object-oriented programming techniques. We use DaSSF *entities* to represent the fidelity-independent components of the simulation and use multiple, alternative DaSSF *processes* connected to these entities to represent the fidelity-dependent behavior these components might have. The topology of the machine defined in the DML specifies how DaSSF *channels* connect the entities. DaSSF *events* move along the channels between entities and are generated and consumed by the processes. Events related to the workload, for example, only move along workload-related channels and events related to the network only move on network-related channels: thus these two parts of the simulation are "unaware" of the level of fidelity currently simulated in the other part. The DML also defines the fidelity levels and supplies fidelity-independent and fidelity-dependent parameters to the entities and processes when they are constructed.

The representation of application workload (shown on the left side of the middle layer in Figure 2) forms an especially important part of the simulation. Our techniques for representing applications and computational workloads at a variety of fidelities are described in Section 4. Each approach addresses tradeoffs between the accuracy of the model and the computing resources required for the model.

The collection of simulation output (at the upper-left in Figure 2) is vitally important for understanding the behavior and performance of the simulated system. Our approach is to permit the collection of information on all events (message sends, packetization, switching, etc.) present in the simulation at the finest level of detail. Because of the potentially voluminous nature of such data, we allow for filtering capabilities so that only data of interest will be collected in a given study. Statistical summaries also provide concise views of system performance and behavior. We are also pursuing visualization of these simulations (at the upper-right in Figure 2). As described in Section 6, we focus on both visualizing the execution of the simulation and on visualizing the performance of the simulated system.

The current *à la carte* implementation comprises low- and medium-fidelity models of a network and low-fidelity and direct execution (nearly perfect fidelity) models of workload. These models support studies of simulation performance and scaling, and also the properties of the simulated systems themselves. Ongoing work in our iterative development approach aims to improve the fidelity of the representations and protocols with validation at each stage. Future work will further emphasize validation, the representation of I/O and storage, and wide-area networking.

3 Describing Machine Architectures

DaSSF models may be constructed from DML files or programmatically. When DML scripts are used, the model topology is defined in the model DML file. Properties of model components, the number to be instantiated, and their connectivity are specified in this file. The machine DML file describes the hardware platform that the simulation will run on. The runtime DML file contains runtime information such as simulation start and end times and the names of the other DML files. DaSSF provides a partitioner that constructs the simulation components from the topology in the model DML file and assigns these components to the parallel computing platform.

For models comprising a large number of components specification in DML quickly becomes cumbersome, running to millions of lines of text for the models in which we are interested. This motivated the development of a tool that generates DML for some space of models.

We have defined a machine representation language suited to very compact representation of particular topological structures in which we are interested, e.g. quaternary fat trees, with special recognition of the components of concern, while preserving the ability to write directly in DML. Details of the design and use of this hardware description language are beyond the scope of this report; the complete description is available elsewhere [9]. Instead, a single example serves to give a sense of the economy of the notation. The text

```
l0=n[4096]
l1=s[1024]
l2=s[1024]
l3=s[1024]
l4=s[1024]
l5=s[1024]
l6=s[1024]

connect l0,l1
connect l1,l2 [1, 0, 2, 3, 4, 5]
connect l2,l3 [2, 0, 1, 3, 4, 5]
connect l3,l4 [3, 4, 0, 1, 2, 5]
connect l4,l5 [1, 0, 2, 3, 4, 5]
connect l5,l6 [5, 1, 2, 3, 4, 0]

model.dml{}
```

expands to approximately 6MB DML ASCII text and fully specifies a model of a 4096-node machine arranged as a six-layer quaternary fat tree, a mathematical description of which may be found elsewhere [10].

4 Representing Workloads

The representation of the application workload is an important component in the *à la carte* framework. The framework is designed to allow multiple representations with varying degrees of fidelity. In this section we describe several workload representations that have been implemented.

The workload on each computational node is represented in the simulation software as a **TSMPNode** instance (a DaSSF *entity*) that encapsulates the attributes and behavior common to all types of workloads. This instance is connected via DaSSF *channels* to a network interface card entity **TNIC** (discussed further in the next section). When the workload sends or receives a message, a **TMessage** instance (a DaSSF *event*) moves between the work-

load and NIC entities on the channel connecting them: the DaSSF discrete-event infrastructure manages the delivery of the event to the appropriate DaSSF *process* at the correct simulated time. The workloads discussed in the rest of this section differ in the implementation of the DaSSF processes connected to the **TSMPNode** instance. We use a factory object, **TWorkloadFactory**, constructed from the DML specification for the simulation run, to manage the instantiation and attachment of the correct workload processes to the workload entity.

In the simplest workload model, the user specifies exactly what messages will be sent from each SMP node in the network topology, the time the message is sent, the destination of the message, the message size in bytes, and optionally, the data content of the message. This workload model is useful for testing specific features of the network models because the content of messages is precisely controlled. It can also be used in trace-driven studies of network behavior.

The *statistical workload* model is characterized by three random variables: an exponentially distributed delay between messages, an exponentially distributed message size, and the message destination, where all possible destinations are equally likely. The average values for message delay and size are specified in the DML model input. The destinations to which each source node can send can be specified individually for each node.

The *ping workload* model was developed to facilitate comparison of the simulation with ping tests conducted on the network hardware. Parameters for this workload are the exact size of the message, the exact delay between messages, the number of messages to send, and the message destination for each message source.

The *direct-execution workload* component provides a means to generate network messages according to the demands of an actual running application. In direct execution simulation the application is executed on the same machine used to perform the simulation. The application is typically modified to call the simulator only for those operations that differ between the host machine and the simulated machine. Using the host machine to directly execute some instructions rather than simulating all instructions can result in considerably faster execution with minimal loss of accuracy when the host and target have similar architectures. Our experiments with direct execution thus far have focused on simulation of communications on the interconnection network and direct execution of the computational aspects of an application.

From time series of fine-grained simulations we are also using learning algorithms to construct *reduced models* of the full system dynamics. This involves regression techniques like neural networks or dimension reduction methods such as the Karhunen-Loeve expansion. When these techniques are more mature they will be implemented within our framework as a new type of workload.

5 Representing Networks

For our first studies rather than modeling the processors and memory hierarchy of an SMP node in detail we chose to emphasize modeling the interconnection network and routing protocol. The ASCI Q machine utilizes a Quadrics interconnection network [2], in which the switches are connected in a quaternary fat-tree topology. The current network models assume this arrangement but our framework is general enough to include other topologies.

The network model contains two types of DaSSF entities, representing the network interface card (**TNIC** instances) and the network switch (**TSwitch** instances). The NIC has an incoming channel for receiving messages from its SMP node and an outgoing channel for sending messages to its SMP node. Additionally, the NIC has an outgoing channel and an incoming channel that connect it to its network switch. Each network switch has 8 incoming channels and 8 outgoing channels. At the level nearest the NICs, 4 of the channels communicate with 4 NICs and 4 communicate with the next level in the quaternary fat-tree. Higher in the fat-tree, communication involves only other switches.

The DML model input is used to create **TNIC** and **TSwitch** instances connected by the DaSSF channels described above with the requested network topology. A **TNetworkFactory** object (also created according to the DML) manages the creation of several DaSSF processes connected to the NICs and switches—which processes are created depends on whether a low- or medium-fidelity network simulation was requested.

5.1 Low-Fidelity Quadrics Network

The *à la carte* low-fidelity network implementation represents the network as a circuit-switched fat-tree network. The source–target patterns for messages can be configured. The switches with four *up* ports and four *down* ports are modeled at the packet level with a simple protocol that allocates a circuit through the network for each message. Using a single circuit for an entire message may be adequate for some types of simulation studies where the details of network traffic congestion are not important. We have run the low-fidelity simulation with a simple statistical workload on a variety of sample models from 1 to 4096 computational nodes (up to 6144 switches in a fat-tree network).

5.2 Medium-Fidelity Quadrics Network

The *à la carte* medium-fidelity network model builds on the low-fidelity model by enhancing its accuracy and realism. This model is much closer to representing actual hardware and mimicking the behavior of network protocols in use on real systems. We expect the resolution of this representation to be sufficient for even the most detailed network studies.

The primary requirement is the ability to accurately model the movement of packets in Quadrics networks consisting of Elan network interface cards [11, 12] connected to Elite crossbar switches [13, 12] in a fat-tree network at nearly *flit* (16-bit unit) resolution. We need to accurately track the movement of the message across the PCI bus between main memory and the network interface card (NIC), account for its packetization, and clock the transfer of data across the network. Because contention may exist in the network, different parts of the packet may move at different speeds through the switches (i.e., buffering and delays may occur anywhere in the network).

Upgrading the low-fidelity network model to the required higher fidelity was straightforward—a direct benefit of careful design. The basic **TSMPNode**, **TNIC**, and **TSwitch** entities remain essentially unchanged. We use object factory methods to generate the appropriate **TRoutingAlgorithm** or **TFlowControl** instances for the fidelity specified in the DML input file. The medium fidelity **TCircuitAlgorithm** and **TCircuitControl** classes are similar to their low fidelity counterparts except for their internal logic, which is considerably more complex, and the type of packets they handle. The design relies on the tracking of the head and tail of the packet throughout its history, along with various flit-level tokens specified in the Elan protocol.

In addition to tracking the movement of the head and the tail of packets through the NICs and switches, we account for the existence of two virtual channels sharing bandwidth at switches (but without age-based priorities, etc.). The Elan *AckNow* request may occur anywhere within the packet (usually after 64 bytes or at the end of the packet). The *EOP_GOOD* tokens free the virtual channels used by the packet. The *START/STOP* tokens are accounted for by buffering of packets at the incoming links to switches if no outgoing virtual channel is available. We model the PCI bus as half-duplex, and account for writes to the NIC's command port. Finally, we allow wildcards for packet routing on upward links. We do not yet model error conditions or the hardware support for broadcast communications. Adding further resolution may not be cost effective because of the uncertainties involved in the performance of the operating system on the node, memory issues, and PCI bus behavior.

The basic strategy for dealing with packets is as follows. When the head of the packet reaches an entity like a NIC, switch, or node, it leaves a reservation at the entity. The head of the packet is forwarded along the route as soon as possible—it might be delayed slightly for switch logic or may be delayed significantly if it is queued for later transmission. As soon as the head leaves the entity, the reservation keeps track of how many bytes remain to be transmitted. The tail of the packet cannot be forwarded until it has been received from the previous stage and the number of bytes remaining at the current stage is zero. The "okay" event proceeds along the reverse path at full speed, and the "good" event cleans up the reservations. There is some fairly complex timekeeping logic for multiplexing the transmission of packets in switches and the receipt of them in NICs.

The studies that have been conducted with this network model are described elsewhere [14]. The models have been calibrated to the behavior of an existing cluster computer of 64 nodes with 256 Alpha/Linux processors connected by an ELAN3 Quadrics network. The calibrated simulation accurately represents real MPI-based pings on the network to within about 750 ns; lower-level pings are modeled accurately to about 100 ns. The network and workload modules have been validated against the real behavior of a representative ASCI application, SWEEP3D. We can predict the execution time of this application to within about 10%, even using the relatively coarse application timers available for our experiments. Our study of the scaling properties of our simulation indicates that it can handle much larger application instances than SWEEP3D.

6 Visualization

Our visualization efforts focus both on viewing the execution of the simulation and on displaying the performance of the simulated system. Visualization also aids in debugging the simulation itself, in developing and evaluating the efficiency of load balancing of the simulation entities, and in understanding synchronization between simulation timelines. Visualizing the simulated system allows end users to understand how varying workload or network architecture affects the overall performance of an advanced or novel architecture. Communication patterns, levels of network usages, and the presence of bottlenecks are all made manifest. Our visualization approaches include direct representations of the architecture as well as innovative abstractions of the architecture and dynamics of the system.

Flatland is an immersive visualization development framework created at the University of New Mexico as part of the Homunculus project [15]. It is used to facilitate rapid prototyping and research in scientific and information visualization, immersive environments and interfaces, and human factors engineering. Utilizing the Flatland framework has leveraged the development of a visualization capability for *à la carte*.

The visualization tool consists of a set of physical representations suitable for representing the topology of the network which in turn is used to register the events generated by the simulator or possibly a real system with proper instrumentation. Several representation have been developed, starting with a suite of direct representations, and followed by more abstract views. The Layered Block view represents the hierarchical fat-tree topology in a variant of a connectivity or adjacency matrix. The self-similar Fractal and H Tree representations take greater advantage of the topology of the network and use 2 dimensional space efficiently. Figure 3 is an example of the H Tree representation with packet flow between switches represented as arcs.

The *à la carte* visualizer currently provides the ability to read the data from a simulation run and run it for-

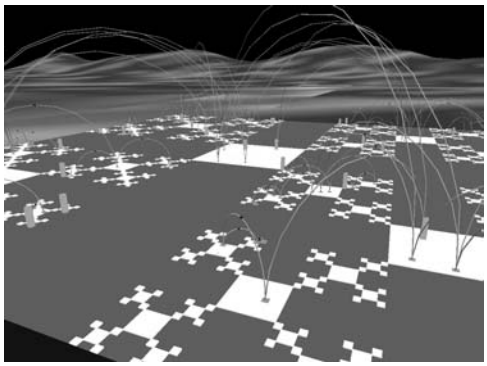

Figure 3. H Tree with packet flow represented as arcs; the white squares represent network switches.

ward and backward in time, controlling the speed of the animation of the events. It also allows the user to toggle on and off the various modes of the representations, such as whether packet flow is displayed as straight lines, arcs, or not displayed.

7 Conclusion

The *à la carte* simulation framework enables the seamless composition of hardware, protocols, and workloads of varying fidelities into a single simulation. The availability of multiple fidelities for each component allows the choice of model to be governed by the requirements of a particular study, thus optimizing the use of simulation time available for the study. Using DaSSF as the underlying parallel discrete event simulation substrate and Flatland as the basis for visualization has enabled the *à la carte* team to focus on the object oriented development of components in the computer architecture domain. We have implemented multiple models for workload and network components and will continue to add to the repertoire. We have also validated the models and conducted studies that illustrate their use; this work is continuing with larger validation, scaling, and case studies.

8 Acknowledgements

This work was carried out under the auspices of the Department of Energy at Los Alamos National Laboratory under ASCI DisCom2. We would like to thank LANL's DisCom project leader, Stephen Turpin, and the *à la carte* project leader, Adolfy Hoisie, for their support.

Thanks to members of the extended LANL *à la carte* team for technical input, including Mike Boorman, Fabrizio Petrini, and Harvey Wasserman; and to David Nicol and Jason Liu (Dartmouth College).

References

[1] `http://www.c3.lanl.gov/~parsim`.

[2] R. Kaufman. The Q Supercomputer and Compaq. `http://www.compaq.com/hpc/news/news\_hpc\_60171.html`, November 2000. High Performance Technical Computing News, Issue 18.

[3] R. L. Bagrodia. Parallel Languages for Discrete-Event Simulation Models. *IEEE Computational Science & Engineering*, 5(2):27–38, Apr-June 1998.

[4] Y.-H. Low, C.-C. Lim, W. Cai, S.-Y. Huang, W.-J. Hsu, S. Jain, and S. Turner. Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation. *Simulation*, 73(3):170–186, March 1999.

[5] Jason Liu and David M. Nicol. *Dartmouth Scalable Simulation Framework User's Manual*. Dartmouth College Dept. of Computer Science, Feb 6 2002.

[6] David M. Nicol and Jason Liu. Composite Synchronization in Parallel Discrete-Event Simulation. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):433–446, May 2002.

[7] James H. Cowie, David M. Nicol, and Andy T. Ogielski. Modeling the Global Internet. *Computing in Science & Engineering*, 1(1):30–38, 1999.

[8] `http://www.ssfnet.org/SSFdocs/dmlReference.html`.

[9] Nick Moss. A Compact Machine Representation Language for Simulation of Large-Scale Parallel Architectures. Technical Report LA-UR 02-6058, Los Alamos National Laboratory, 2002.

[10] Brian W. Bush. Idealized Q Layout. Los Alamos National Laboratory, 2002.

[11] Quadrics Supercomputers World Ltd., Bristol, UK. *Elan Reference Manual*, January 1999.

[12] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.

[13] Quadrics Supercomputers World Ltd., Bristol, UK. *Elite Reference Manual*, November 1999.

[14] Kathryn Berkbigler, Graham Booker, Brian Bush, Kei Davis, and Nicholas Moss. Simulating the Quadrics Interconnection Network. Submitted to *High Performance Computing Symposium 2003*, March 2003.

[15] `http://www.ahpcc.unm.edu/homunculus/indexold.html`.